Development of Online Course Application with Ruby on Rails framework

Major project report submitted in partial fulfillment of the requirement for the degree of Bachelor of Technology

in

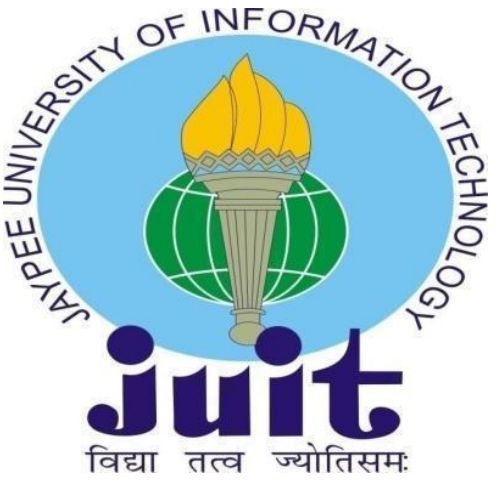
Computer Science and Engineering

By

ANURAG KANAUJIA (181363)

UNDER THE SUPERVISION OF

Dr. RAVINDARA BHATT



Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, 173234, Himachal Pradesh, INDIA

DECLARATION

We hereby declare that this project has been done by us under the supervision of **Dr. Ravindara Bhatt, Associate Prof, Department of CSE** Jaypee University Of Information Technology. We also declare that neither this project nor any part of this project has been submitted elsewhere for the award of any degree or diploma.

Supervised by:
Dr. Ravindara Bhatt
Associate Prof
Department of Computer Science & Engineering And Information Technology
Jaypee University Of Information Technology

Anurag Kanaujia
(181363)
Department of CSE
Jaypee University of Information Technology

CERTIFICATE

This is to certify that the work which is being presented in the project report titled **"Development of Online Course Application with Ruby on Rails framework"** in partial fulfillment of the requirements for the award of the degree of B.Tech in Computer Science And Engineering and submitted to the Department of Computer Science And Engineering, Jaypee University of Information Technology, Waknaghat is an authentic record of work carried out by **"Anurag Kanaujia (181363)"** during the period from February 2022 to May 2022 under the supervision of **Dr. Ravindara Bhatt**, Department of Computer Science And Engineering, Jaypee University of Information Technology, Waknaghat.

Anurag Kanaujia
(181363)

The above statement made is correct to the best of my knowledge.

Dr. Ravindara Bhatt
Professor
Computer Science & Engineering And Information Technology
Jaypee University of Information Technology, Waknaghat, India
Dated:

ACKNOWLEDGMENT

First We express our heartiest thanks and gratefulness to Almighty God for His divine blessing makes us possible to complete the project successfully.

We are grateful and wish my profound indebtedness to **Dr. Ravindara Bhatt**, **Associate Prof**, Department of CSE Jaypee University Of Information Technology, Wakhnaghat. Deep Knowledge & keen interest of my supervisor in the field of "**Development of Online Course Application with Ruby on Rails framework**" to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts, and correcting them at all stages have made it possible to complete this project.

We would like to express my heartiest gratitude to **Dr. Ravindara Bhatt,** Department of CSE, for his kind help to finish my project.

We would also generously welcome each one of those individuals who have helped us straightforwardly or in a roundabout way in making this project a win. In this unique situation, We might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, We must acknowledge with due respect the constant support and patients of my parents.

Anurag Kanaujia
(181363)

ABSTRACT

This thesis discusses developmental steps for web-based courses, an app that allows the user to log in, Subscribe to the tutorial and see its chapter on smart content similar to what Udemy does with some additional features and support. Describes in detail the implementation of the application at the end and back. Thesis is possible designed and explained how to improve a web application using Ruby on Rails technology.

TABLE OF CONTENTS

DECLARATION	i
CERTIFICATE	ii
ACKNOWLEDGMENT	iii
ABSTRACT	iv
INTRODUCTION	3-7
LITERATURE REVIEW	8-16
SYSTEM DEVELOPMENT	17-41
PERFORMANCE ANALYSIS	42
CONCLUSION	43-44
REFERENCES	46

LIST OF FIGURES

Fig. 2.1 The Model View Controller architecture 9

Fig. 2.2 Architecture of standard Ruby on Rails web application 12

Fig. 3.1 Visual overview of a Three-tiered Web Application 19

Fig. 3.2 High-level view of the Online Course application. 20

Fig 3.3 Model-View-Controller Architecture of the online course application 21

Fig 3.4 DB design for online course application 25

Fig 3.5 POST search/topics 30

Fig 3.6 POST /topic/:slug/courses 31

Fig 3.7 POST search/courses 32

Fig 3.8 GET course/:slug/modules 33

1. INTRODUCTION

The online course application aims at supporting application users in learning and achieving numerous skills by providing different course contents and a user friendly interface. The application interacts with the users through a website, which is always available via the Internet. The project is divided into two functional blocks: the application front-end and the application back-end.

Application Front-end The front end of the online course application is the user interface. The website page is the application interface through which users can interact with the course. You can log in using your email address and app password. You can register for courses, study different chapters, watch videos, and take notes on videos.

Application Back-end The backend of the application consists of a relational database with various database tables where user data is stored and from which user data is retrieved. When a community user submits a request, the request value is validated, structured on the front end, and sent to the back end. The database processes these values and responds to the user interface.

1.1 Problem Statement

In the 21st century, education has undergone new changes with the advancement of technology. The way we read has also changed, so the way we read has changed a lot. Technology is being introduced into the education system and distance learning, online classes and online courses are becoming the norm. There are various forums today that offer many options to learn about a variety of topics. Some of them are free, others require a subscription to access the full curriculum. Improved online forums that have emerged over the past few years have led to improved levels of learning.

Online courses allow anyone to sit down and learn a new skill or hone an existing one. Learning new skills expands your repertoire and improves your professional skills. Employers will appreciate if you take the time to hone your skills and keep up with the times. Studying at home with an online course has its advantages.

Online courses can not only benefit students but also the working professionals,
They can be extremely beneficial for them as they want to learn new skills which are either for their career growth, personal development or just for fun and don’t want to commute.

1.2 Objectives

The goal of this project is to develop and implement a prototype online tutorial using Ruby on Rails technology. The visual user of the application is responsible for managing all user requests, filtering input, submitting requests to the website, managing feedback from the website, filtering responses, and sending web interactions to the user. This is the main feature of the report and we will focus on developing web applications using the Ruby on Rails framework.

The Target Audience

The intended audience for this report is students and practitioners. The concept of online learning allows students to learn from home or any other convenient location. They can find learning materials on the Internet. Textbooks for online learning can be texts, audio recordings, notes, videos, and images.

1.3 Organization

About InterviewBit/ Scaler

Scaler Academy is an online technical college for the world's leading software developers. They offer an in-depth six-month computer science course in the form of live lessons taught by top tech experts and subject matter experts. Well-planned programs enhance the skills of software professionals by providing a state-of-the-art curriculum that uses the latest technologies. This is an interview bit product.

Founders

Scaler Academy (and InterviewBit) was founded by IIIT Hyderabad graduates Anshuman Singh and Abhimanyu Saksena.

Anshuman previously worked at Facebook, leading the team, creating and measuring Messenger features. Anshuman is also part of a team of four that will set up Facebook's London office. He competed and made it to the ACM ICPC World Finalist twice. Abhimanyu led the team that designed the entire New York market for Fab.com. A longtime businessman, Abhimanyu started his first business while in college.

What do they do

Scaler Academy is an online acceleration program that successfully develops the skills of software programmers. We educate students on all the relevant skills needed in the software industry, advise on the recruitment process and align with the best software industry employment opportunities around the world. Academy students spend an average of 35 hours each day on the platform learning and practicing their skills. Give students access to more than 400 professionals from top software companies such as Facebook, Amazon, Google, Directi, and Microsoft, who are active as educators, mentors, teaching assistants, and career coaches. Prominent executives such as Bavin Turakhia (Co-founder and CEO of Directi Group and Zeta), Rajan Anandan (former Google Head of APAC), and Jason Goldberg (CEO of Fab.com) are among the individuals who guide scaler students. It is a department.

Why do they do

InterviewBit does little to help young professionals improve their opportunities to work with leading technology companies. Since 2015, we have transformed the lives of more than 4,000 young professionals by helping leading technology companies find their dreams. Currently, more than 600 companies outsource the recruitment of professionals. Most of the people we deal with have seen early on that they can accomplish even greater things if given the right guidance. What was lacking was not only teaching new skills, but also access to teachers and mentors to guide their careers. For this reason, we launched the Scaler Academy in April 2019. We are confident that talented people will be able to realize their dreams by bridging the gap between their theatrical knowledge and their thematic skills. And the Scaler Academy did just that.

2. LITERATURE REVIEW

2.1 Software Tools Overview

This chapter will provide a brief overview of some of the abbreviations and keywords, namely will be used continuously in the report. The information included in this section is intended to makes reading this report easier for the reader to follow

2.1.1 Web application

A web application (web application) is an application program that is stored on a remote server and served over the Internet via a browser interface. Web services are, by definition, web apps, and many, if not all, websites include web apps. According to Jarel Remick, the Web.AppStorm editor, any website component that performs a function for a user is considered a web app.

Web applications can be designed for a variety of purposes and can be used by anyone. From organization to individual for a variety of reasons. Commonly used web applications include webmail, online calculators, or e-commerce stores. Some web apps can only be accessed from certain browsers. However, most are available regardless of browser.

2.1.2 Online community

An online community or community that is visible to a group of people in particular use social media such as email, an internet communication service or instant messaging rather than face to face, social, professional, educational or other purposes. Today visible communities and the internet have become a
a method of communication between people who share the same businesses
interests, hobbies or friendships.

2.1.3 Web Framework

A web application (web application) is an application program that is stored on a remote server and served over the Internet via a browser interface. Web services are, by definition, web apps, and many, if not all, websites include web apps. According to Jarel Remick, the Web.AppStorm editor, any website component that performs a function for a user is considered a web app.

Web applications can be designed for a variety of purposes and can be used by anyone. From organization to individual for a variety of reasons. Commonly used web applications include webmail, online calculators, or e-commerce stores. Some web apps can only be accessed from certain browsers. However, most are available regardless of browser.

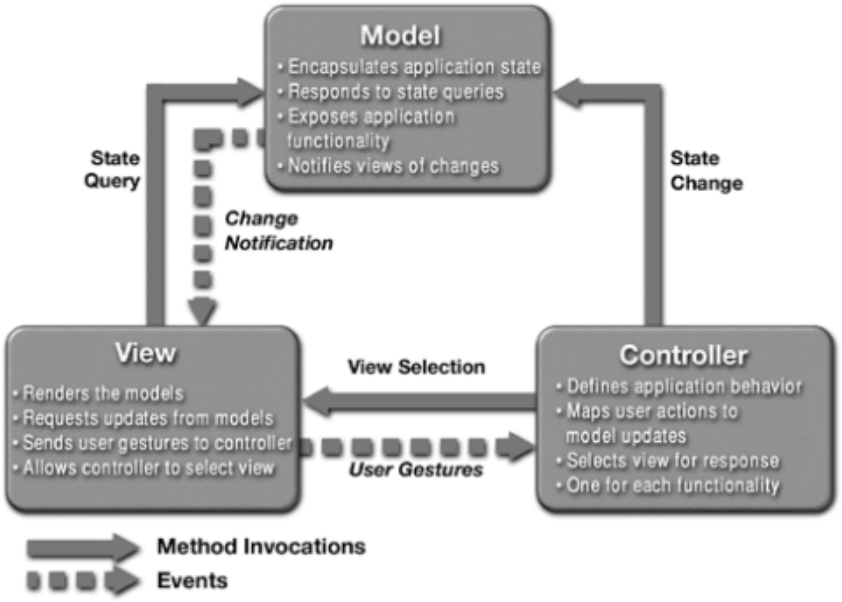


Fig. 2.1 The Model View Controller architecture

2.1.4 Don't Repeat Yourself

The Don't Repeat Yourself (DRY) principle states that logic duplication should be eliminated by abstraction. Process duplication should be eliminated by automation. Duplication is useless. Adding unwanted code to your code base increases the amount of work required to extend and maintain your software in the future.

Duplicate code contributes to technical debt. Whether the duplication is due to copy-and-paste programming or a lack of understanding of the application of abstractions, the quality of the code is degraded. If it can be automated, process duplication is also wasted. Manual testing, manual builds and integration processes should be eliminated using automation wherever possible.

2.1.5 Convention over Configuration

Convention over configuration relies on developing programs using native procedures, functions, classes, and variables in the underlying language. This approach reduces or eliminates the need for additional software configuration files, ultimately making software development, code consistency, and maintenance easier and faster. However, to follow these rules, software developers must be familiar with the underlying framework.

Software frameworks that support the convention approach to configuration development include Ruby on Rails, JavaBeans, and CakePHP.

2.1.6 Representational state transfer

REST (REpresentational State Transfer) is an architectural style for providing standards between computer systems on the Web, making it easier for systems to communicate with each other. REST-compliant systems, often referred to as RESTful systems, are stateless and are characterized by the fact that they separate the interests of clients and servers. Explain the meaning of these terms and why they are useful properties for services on the web.\

Separation of Client and Server

In the REST architectural style, the client implementation and the server implementation can run independently without knowing each other. That is, client-side code can be modified at any time without affecting server-side operations, and server-side code can be modified without affecting client-side operations.

As long as each side knows the format of the message it sends to the other, they are modularized and kept separate. By separating UI issues from data storage issues, you can increase UI flexibility across platforms and improve scalability by simplifying server components. In addition, the separation of each component allows them to be developed independently.
By using the
REST interface, different clients reach the same REST endpoint, perform the same action, and receive the

same response.

2.1.7 Ruby

Ruby is an open source object-oriented scripting language invented by Yukihiro Matsumoto in the mid-1990s. Unlike languages such as C and C ++, scripting languages do not communicate directly with the hardware. It is written to a text file, parsed by the interpreter and converted into code. These programs are generally procedural and read from top to bottom.

On the other hand, object-oriented languages break down pieces of code into objects that can be created and used as needed. These objects can be reused in other parts of the program and in other applications.

Yukihiro wanted to develop a scripting language that uses object-oriented programming to increase code reuse and speed up development. And the Ruby programming language was born. It uses simple languages and syntax to process data and logic and solve problems.

2.2 Ruby on Rails Framework

Independent programming languages which doesn't include high-level programming also includes ruby. Without rails, ruby is incompatible.

Ruby on Rails is an application platform that sparks interest in Ruby and makes it popular and a compatible with cloud computing

By many great people at ruby onrails.org, Ruby on Rails is "an open-source web framework optimized for programmer happiness and sustainable performance".

Ruby on Rails platform has pre-built Ruby code for communication, file management, database connections, and more. Take care of boring problems so you can focus on solving them. One of the key concepts in Rails is Don't Repeat Yourself (DRY), which is central to the effectiveness of the framework.

There are over 1 million websites written in Ruby on Rails. There is a wide range of powerful business and

entertainment sites such as GitHub, Twitch, Bloomberg, SoundCloud, Hulu, Square, Basecamp, Airbnb, Hulu, The Weather Channel, Instagram, and Twitter. Rails are organized around Model, View, and Controller architecture. The benefits of MVC are to differentiate the business mindset from the user interface, save the ZOMA application code and make it clear where different types of code belong to keep the application easy. Figure 2.2 shows the typical design of the RoR web application.

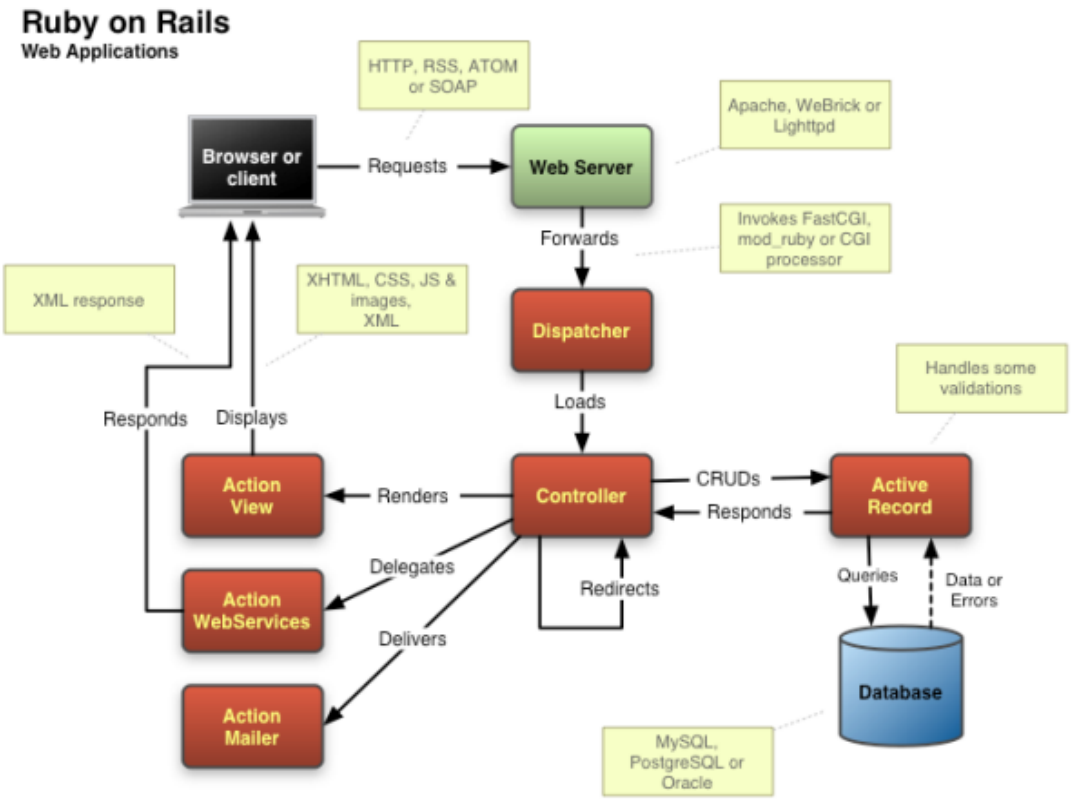


Fig. 2.2 Architecture of standard Ruby on Rails web application

2.2.1 Models

This is called the lowest level and means that you are in charge of data maintenance. It's basically data

because it's logical to it. Since the model is actually connected to the database, it does anything it does with the data. Adding or retrieving data is done in the model component. Since the controller does not interact with the database itself, it responds to requests from the controller. The model interacts with the database and passes the necessary data to the controller. memo. The model does not interact directly with the view.

2.2.2 Views

Data is presented as view components. You actually create a user interface or user interface. So when you think about the view component of your web application, think about the html/css part. Views are created with data collected from model components, but this data is not collected directly, but through controllers, so the view only communicates with the controller.

2.2.3 Controllers

A controller is a component that provides a link between a view and a model and is known as an assistant because it acts as an intermediary. The controller doesn't have to worry about handling the data logically, it just tells the model what to do. After receiving data from the model, the model processes the data, extracts all this information, and sends it to the view to tell the user how to render it. memo. Views and models cannot speak directly.

2.2.4 Ruby on Rails Components

ROR has dozens of attachments for developing web apps. The basic structure of the ROR platform.

2.2.5 Action Controller (ActionController::Base)

The action controller is the part of the Rails application that manages the controller. The activity management framework handles incoming requests from Rails applications, retrieves parameters and passes them to the intended task. Actions are defined as public routes for the controller that are automatically provided to the web server using Train Routes. The two main archetypes used for activity management are search, display, and redirect. Many activities are variations on these themes. Services provided by Action Manager include session management, template rendering and redirect management.

2.2.6 Action View (ActionView::Base)

activity view contains a view of your Rails app. It is able to automatically generate markup outputs like html and xhtml. Action View contains basic templates and nested compressed templates, and also have built in AJAX support. There are three ways to create a task view template. Templates that use a combination of inline Ruby (ERb) code and HTML code have a .html.erb file extension. Templates that use the Jim Weirich Builder::XmlMarkup library have either a .builder or .rxml file extension. Templates that use the ActionView::Helpers::PrototypeHelper::JavaScriptGenerator module have a .rjs file extension.

Active Record (ActiveRecord::Base)

It is a highly-featured Rails Object Relational Mapping (ORM) component that requires no configuration. Naming and conventions are important to maintaining simple and minimal code for defining classes that are stored in database tables.

2.2.7 Action Mailer (ActionMailer::Base)

Action Mailer is a framework for creating email resources. Action Mailer can be used to send emails based on dynamic templates or to receive and process incoming emails.

2.2.8 Active Resource (ActiveResource::Base)

Connect a business object to a Representational State Transfer (REST) web service. With ActiveResource, you can easily expose your ActiveRecord model using REST with very little code. This is a convenient way to create an API without much effort.

2.2.9 Active Support (ActiveSupport::Base64)

A set of useful utility classes and standard extension libraries in Rails. This extension can be useful for many Ruby projects.

2.3 Ruby on Rails Directory Structure

The Rails frame takes on a specific operating time frame. Here is a sample of the high-quality indicators produced in the creation of the railway project.

app-Organize application components. There is a subdirectory containing views (views and helpers), controllers (controllers), and backend business logic (models).

app / controllers-Rails looks for the controller class in the controller subdirectory. The controller handles web requests from users.

app / helpers-helpers The subdirectory contains all the helper classes used to support the Model, View, and Controller classes. This allows you to keep your model, view, and controller code small, focused, and organized. The

app / models-models subdirectory contains classes that model and wrap the data stored in the application's database. In most frameworks, this part of the application is very tedious, long, verbose, and error-prone. You can easily do it with Rails! The

app / view-views subdirectory contains view templates for entering data from the application, converting it to HTML, and returning to the user's browser.

app / view / layouts-Contains template files for layouts used in views. It models the usual header / footer method of wrapping a view. In the view, use layout: default to define the layout and create a file named default.html.erb. Internal by default

Components-This directory contains small standalone applications that bundle components, models, views, and controllers.

config-This directory contains a small amount of configuration code needed by your application, such as database configuration (in database.yml), Rails environment structure (environment.rb), routing of incoming web requests (routes.rb), etc. I am. .. You can also use the files in the Environments directory to customize the behavior of the three Rails environments for testing, development, and deployment.

db-Rails applications typically have model objects that access relational database tables. You can manage

your relational database using a script that you create and place in this directory.

doc-Ruby has a framework called RubyDoc that can automatically generate documentation for your code. You can support RubyDoc with comments in your code. This directory contains all rail and application documentation generated by RubyDoc.

lib-Place the library here unless the library specifically belongs to another location (such as a vendor library).

Logs-Error logs are stored here. Rails creates scripts to help you manage various error logs. Server (server

public-Like the web server's public directory, this directory contains unchanged web files such as: B. JavaScript files (public / javascripts), graphics (public / images), stylesheets (public / stylesheets), and HTML files (public).

Scripts-This directory contains scripts for launching and managing various tools used by Rails. For example, there is a script to generate code (Generate) and a script to start a web server (Server).

Tests-All the tests you create and Rails create are here. The subdirectories of Mocks, Units, Fixtures, and Functionals are displayed.

tmp-Rails uses this directory to store temporary files for intermediate processing.

Vendors-Library provided by third parties (such as security libraries and database utilities beyond the base Rails distribution) can be found here.

README: This file contains basic information about the Rails program and a description of the directory listed above.

Rakefile: This file is similar to Unix Makefile which helps with creating, packaging, and scanning Rails code. This will use the rake utility provided as well as the Ruby installation.

3. System Development

In this chapter the main requirements that the project has to fulfill are pointed out. The possible software solutions are presented. The software needed for the project implementation is discussed and reasons for choosing it is given. A discussion of the hardware requirements is made as well.

3.1 Project Analysis

The desire to design an online course web application is important because Online education enables the teacher and the student to set their own learning pace, and there's the added flexibility of setting a schedule that fits everyone's agenda. As a result, using an online educational platform allows for a better balance of work and studies, so there's no need to give anything up.

Our Project targets students as well working professionals, it can contain articles, videos, quizzes and many more things in order to engage students to try to aim for a better learning environment just like what colleges or universities does but in a more friendly manner.

Students can interact with TA (teaching assistants) if they come up with a new
ideas or doubts meanwhile they can track the progress of their course.

Website is Divided into 4 major parts which articles, challenges, videos
that may contain inside a course which will be inside our chapter which will be belonging to particular course for example python is a course inside python there will be chapters like loops, if/statements and inside chapters there will be all these articles, challenges, videos lets name them entities for sake of generalization.

We will be following the best system design and structure and will remove all the redundant and unnecessary data if they are not used. A user is able to track their progress of what course he/she watches for a better and comfortable viewing experience.

3.2 Project Functional Requirements

Functional requirements describe the desired functions of the project, they are the core of each project and they have an important role in the success of the project. This subchapter starts with a discussion of the main functional needs and continues with explaining each of them in detail. As a small web course platform the

project needs to implement some basic features. In order to achieve a social network functionality the application has to include user roles, profiles, semi-persistent public commentary on the profile, messaging system and a traversable publicly articulated social network displayed in relation to the profile. Including all these features in a web application transforms this application to a basic social network service. During the project development additional features can be added on to expand the application functionality. In the following section the main features are listed and discussed.

3.2.1 User Role.

The project application requires different types of users, where user role corresponds to the privileges given to a certain user. Based on the application role users can be divided into two categories: superusers - administrators, and normal users. Superuser is a user with extended administrative privileges. He has control over the users in the social network. Superuser can delete, suspend or activate a normal user. Administrator users can access every user's profile, delete messages and Requirements and photos with inappropriate content. Normal user is a user, a member of the learning community, who can modify only his/her profile and has no administrative privileges. A small community platform has to implement different user roles. An online course application needs to have administrator users, who have control over the network and can make decisions about users and their profile's content. Without user roles a small community can hardly exist and will be very difficult to maintain.

3.3 Project Design Requirements

This project follows a three-phase client software development model. The app is divided into three sections called "tiers". Categories are usually separate machines. The three categories include the customer category, the application category and the website category. Client sends HTTP requests via a web browser. Web server services request by performing queries and updates against the database server. The results are then reverted to a web server, where the HTML visual effects are generated. The HTML template is restored to the client browser. Figure 4.1 shows the structure of the three phases of the project.

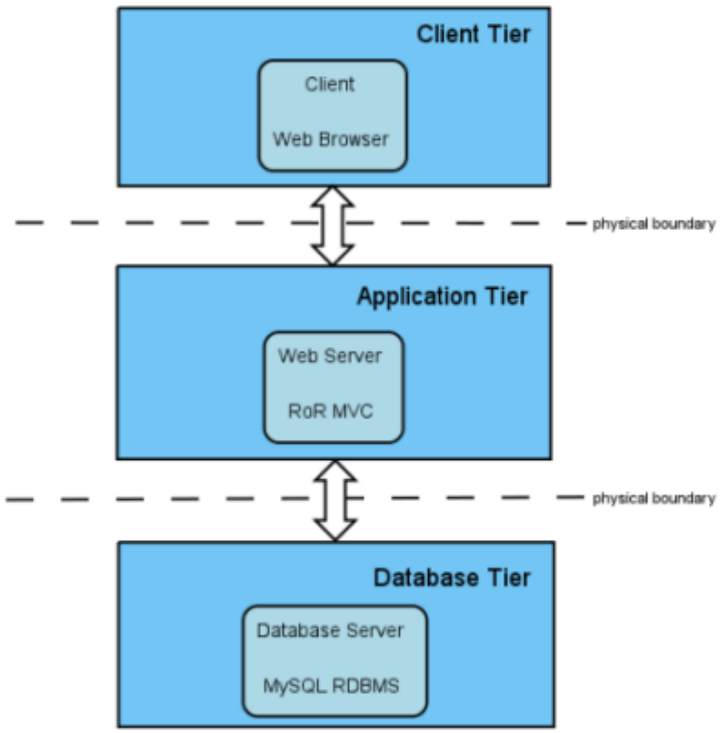


Fig. 3.1 Visual overview of a Three-tiered Web Application

Client category - the first end of the application is a web browser. Each client has its own computer with installed operating system. The web browser is running on top of the operating system. Common operating systems in this category include various Linux distributions such as Ubuntu, Red Hat and SUSE, Microsoft Windows and

Mac. Typical web browsers include Mozilla Firefox, Internet Explorer, SeaMonkey, Opera, Safari.

Web Application Tier - intermediate section covers the business concept of the application. Controls application performance by performing detailed processing. In the MVC architecture of this application, the Application-tier is used as the viewer, controller and model layer.

Database Tier - the back end of the application contains MySQL RDBMS running on a stored server / local server. This section stores and downloads information. The website section keeps data neutral and independent of the business intelligence.

During the development of the Online course application, front-end and back-end are available on different local servers.

After making it clear about the functional, software and hardware requirements of the project, and having understood the idea of the Model-View-Controller architecture, the next step is to design the web application architecture.

3.3 Online Course Application Architecture

The application is divided into application front-end, which is the user interface, from where the user interacts with the community platform, and application back-end, which is the database system of the platform.

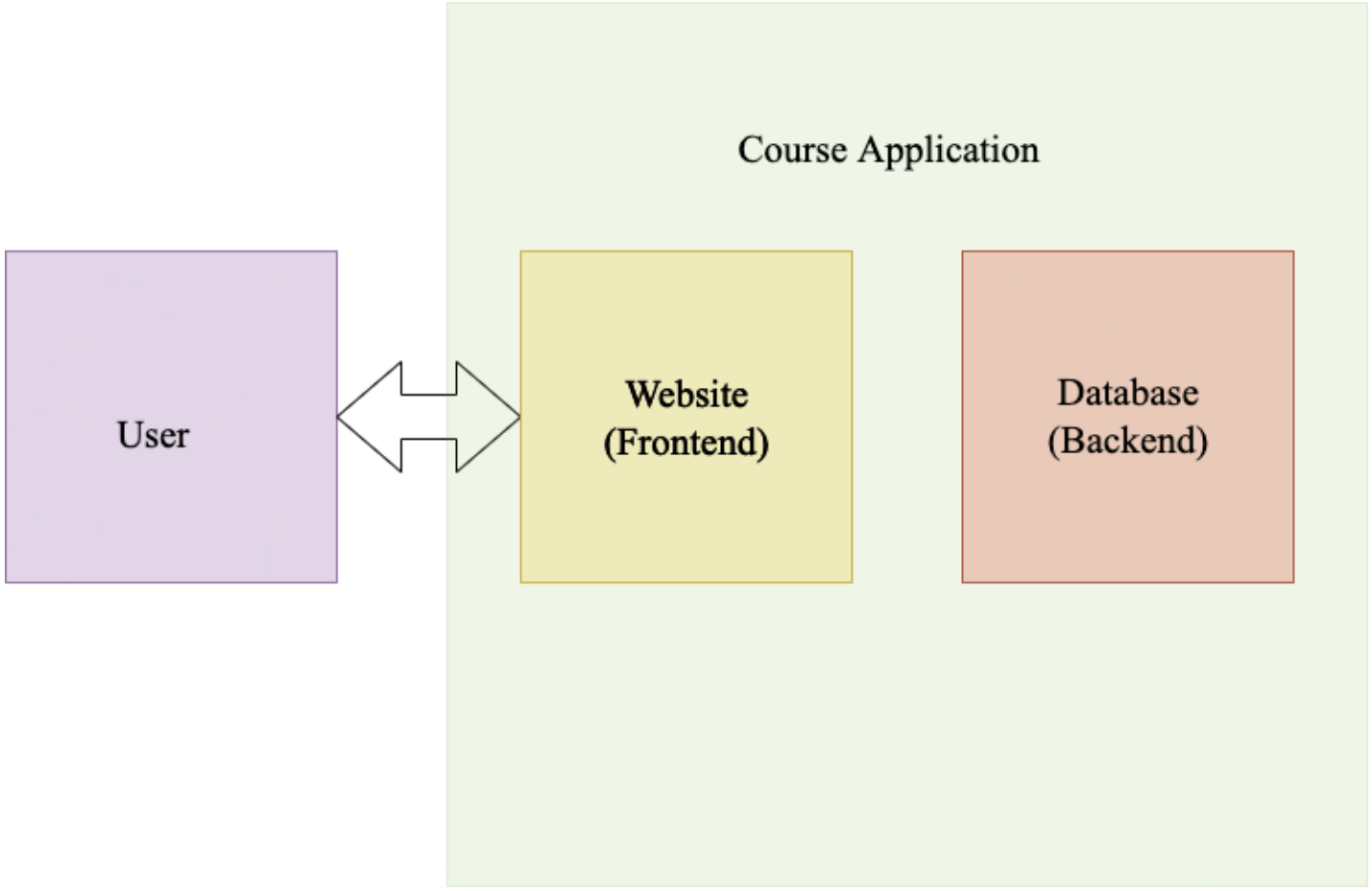


Fig. 3.2 High-level view of the Online Course application.

Once the functional requirements are partitioned into actions of requests and responds between users and application, the next step is to identify the business logic, presentation logic, control flow logic and model them as objects. The following sub chapter introduces the general MVC structure design of the Web Course application and then we will go further into internal design of each layer of the MVC to decide which objects each layer needs.

3.4 Overview of the Application MVC Architecture

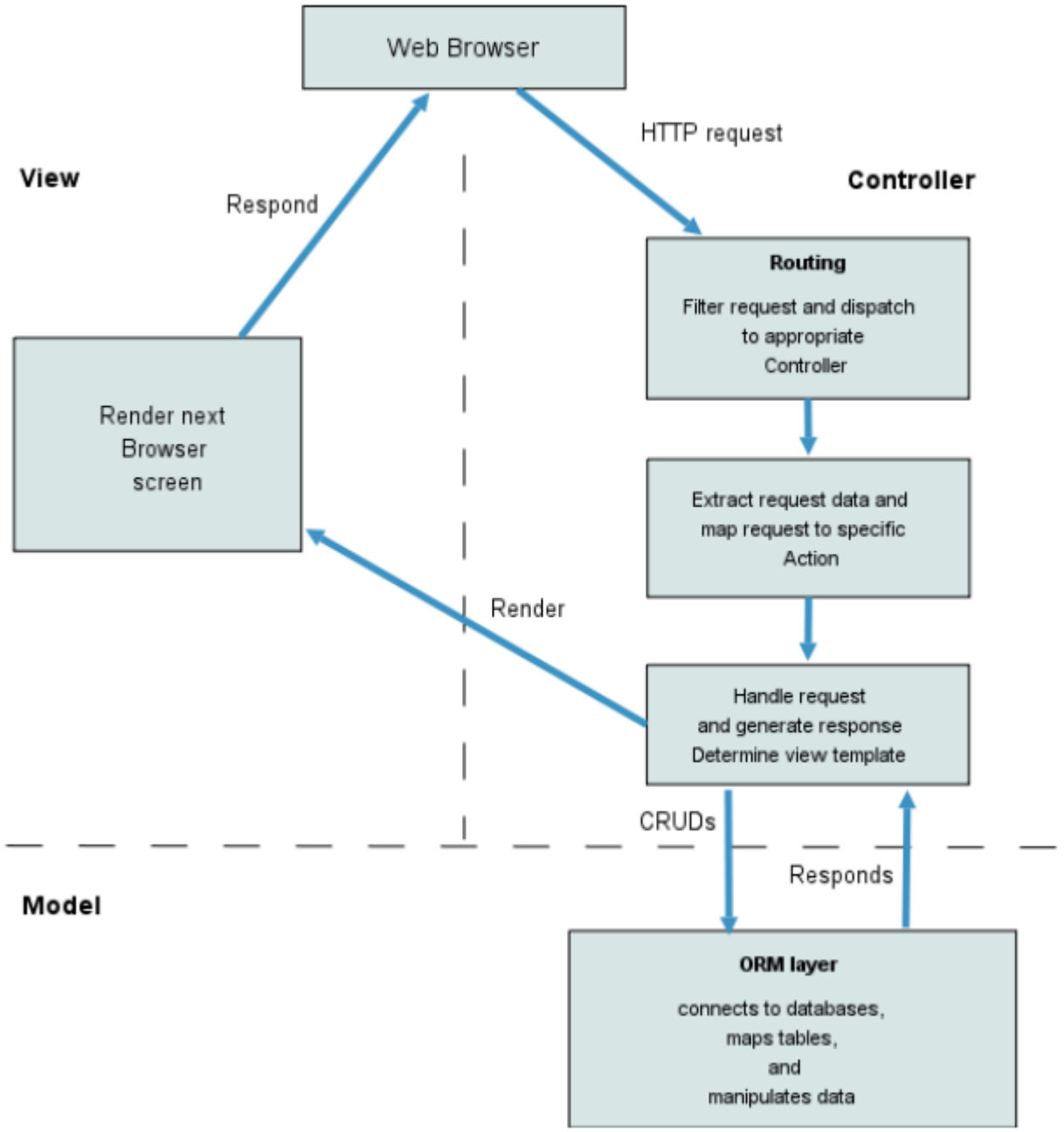


Fig 3.3 Model-View-Controller Architecture of the online course application

A user sends a HTTP request to the controller through the web browser. The controller receives the request, handles the request with corresponding control logic for this specific request, and invokes a view to respond to the user browser. The model has different objects to store different data type and objects which have business rules to process the data. When receiving request data from a browser, data is checked for invalid input value, refined and sent to the back-end. The response data from the back-end is as well refined and sent to the web browser. The view is responsible for generating the user interface, normally based on data in the model. The view includes different HTML templates and layouts, their dynamic values are the values stored in the value objects. The object state and value can be updated.

3.5 Implementation of the Application MVC Architecture

3.5.1 The View Layer

The Ruby on Rails technology for handling user view in this online course application are rhtml templates. Rhtml templates are a mixture of HTML and embedded Ruby (ERb). They are used to generate HTML pages. In general, this application uses ERb for presentation of the dynamic contents and logic for presenting the view.

There are some considerations in the view layer

- Views only have view logic.
- Inline Ruby code remains at least
- Using View Template Assistant
- Using Partial View Templates and Layouts
- View-friendly user interface

The control layer is responsible for the control logic and the model layer is defined to control the business rules. For the background view, focus only on the logic of the presentation. Rails also has many built-in helper modes and modules that you can use to simplify templates and incorporate complex presentation logic. Many online tutorial pages share the same top, tail, and sidebar. You can see the same performance in many places. Common parts and components are used to avoid duplication of design. The online course app uses two editing styles. What registered users use is a top navigation and search bar, a center where all templates are loaded, a sidebar on the right, and a footer at the bottom.

3.5.2 The Model Layer

The model layer integrates the data elements and the business concept of the application. The object-oriented map (ORM) layer of the application. The project closely follows the standard ORM model: a map of classrooms, rows to objects and columns in object attributes

3.5.3 The Controller Layer

MVC control layer is used for application control alignment. Controlller works with both model and view layer. For the view background, the controller is responsible for selecting the next view. For the model background, the controller will apply the assets to the business value and apply the business mindset processes. For its simplicity, the web application receives an incoming request from a web browser, processes it, and sends feedback. The controller acts as a single interface for all incoming HTTP requests from clients. The information is encrypted in the application URL, a sub-system called “route” is used to determine what to do with that request. The web application determines the name of the controller responsible for this particular application, as well as the list of any other application parameters. Usually one of these additional elements indicates the action to be applied to the targeted controller. Once the controller has been identified, a new instance of the control panel is created, and its process is called, to transfer the request details and response item. The controller then calls the path with the same name and action. The course of action in the controller will require you to process business idea, review business value items and determine if the next page is back in the browser. The config / routes.rb file contains all the application route information. The Route section draws a map that links the external URLs of the web community application. The routes.rb file is processed from top to bottom when the application is received. The request will be sent to the same original route. If you do not have the same route, HTTP 404 status - An error message not found, is returned to the caller. To achieve the required functionality of the RESTful app, Named Routes and Nest was used. A look at the config / routes.rb file will give the reader a brief overview of the different types of routes.

Once the route has determined which controller to use the application, the controller is responsible for making the application concept and producing the appropriate output. The controller ensures that the model data is available for viewing, so we can display this data to the user, and the controller is responsible for storing and updating user data on the model.

3.6 Back-end - Database Design

After introducing the MVC structure design of the Web Community application, it is time to have closer look on the database design. Developing with Rails provides a convenient way of creating, altering and managing databases and tables by using migrations. Migrations are sub classes of ActiveRecord::Migration class and can be found in the db/migrate directory. Creating a model class in Rails application, automatically generates a create table migration class with two class methods: up – for performing the required transformations and down – for reverting them. For the back-end of the Web Community project a MySQL RDBMS is used. The design of database is strongly connected to the design and the business rules of the application's model layer. The configuration of the application's database is done in config/database.yml file. Data stored and retrieved from tables on the database server are as follows.


videos
id bigint
title string
created_at datetime
updated_at datetime
published_at datetime
url string
slug string
description text
duration float8
thumbnail_url string
subtitles_url string
user_course_tracking
id bigint
course_id bigint
chapter_id bigint
user_id bigint
record_id bigint
record_type string
timeElapsed float8
status status
progress_percentage float8
notes
id bigint
user_id bigint
record_id bigint
record_type string
created_at datetime
updated_at datetime
content text
video_notes
id bigint
note_id bigint
title string
at_time float8
courses
id bigint
name string
article_collabrators_id bigint
slug string
description text
requirements text
suitable_for text
created_at datetime
updated_at datetime
published_at timestamp
cover_img_alt string
chapter_entities
id bigint
chapter_id bigint
record_type string
record_id bigint
order integer
created_at datetime
updated_at datetime
user_course_registrations
id bigint
course_id bigint
user_id string
status bigint
created_at datetime
updated_at datetime
chapters_owner
id varchar
owner_id bigint
owner_type varchar
chapter_id bigint
order integer
created_at datetime
updated_at datetime
chapters
id varchar
title varchar
slug varchar
created_at datetime
updated_at datetime
dbdiagram.io

Fig 3.4 DB design for online course application

courses

This table holds all the information of what a course can have

- name - Name of the course
- article_collaborators_id - Instructor of the course
- slug - keyword of course
- description - description of the course
- requirements - prerequisites of the course
- published_at - published date for the course
- cover_image_thimbnail - thumbnail image
- cover_image - cover image of the course
- suitable for - for which the course is for

chapters

This table holds all the information of what a chapter can have which will come inside a course

- title - Name of the chapter
- slug - keyword of chapter

chapter_entities

This table holds all the information of what a course must include its a generalized mapping table for joining multiple things named entities to the course itself

- chapter_id - which chapter it belongs to
- record_type - which entity it is mapping for
- record_id - id for that entity
- order - in which order it will come

chapter_owners

This is a mapping table to course to chapter

- chapter_id - which chapter it belongs to
- owner_type - which entity it is mapping for it can be topic, hub or anything
- owner_id - id for that entity
- order - in which order it will come

videos

This table holds all the information of what a video must include

- title - Name of the video
- duration - duration for the video
- slug - keyword of video
- description - description of the video
- published_at - published date for the video
- cover_image_thimbnail - thumbnail image
- url - url for the video in which it is deployed
- thumbnail_url - thumbnail for the video
- subtitle_url - subtitle for the video

notes

This table holds all the information of what a note must include which will be taken in between watching the video, article or anything

- user_id - which chapter it belongs to
- record_type - for which entity it is storing notes
- record_id - id for that entity
- content - content of the notes

video_notes

This table holds all the information of note can have which will be taken from a video

- note_id - mapping from notes table
- at_time - duration for the time taken in notes
- title - title for that particular video

user_course_registerations

This table holds all the information user for which he/she has a subscribed for course

user_course_trackings

- user_id - id for that user
- course_id - course_id for which he/she is still pursuing
- status - status is an enum (0,1), 0 for not completed and 1 for completed

3.7 Implementation of the API

Our project basically have four methods defined below

Method	Description
GET	**Retrieve information about the REST API resource**
POST	**Create a REST API resource**
PUT	**Update a REST API resource**
DELETE	**Delete a REST API resource or related component**

There are basically ten guidelines that you can follow to make your API endpoints better:

1. Use nouns.
2. Use intuitive, clear names.
3. Use lowercase letters.
4. Avoid special characters.
5. Use forward slash (/).
6. Separate words with hyphens.
7. Don’t use file extension.
8. Use camelCase for parameters.
9. Use API versioning.
10. Consistency.

Due to the confidentiality of the code and API I will be changing some endpoints and data

- POST search/topics

```
"data": [
       {
           "id": "63",
           "type": "topics",
           "attributes": {
               "title": "Topic-18",
               "slug": "Topic-Slug-18",
               "description": "Ad inventore sequi. Aut magnam soluta. Sus
               "published_at": "2022-05-16T16:37:37.000Z",
               "created_at": "2022-05-16T16:37:37.162Z",
               "updated_at": "2022-05-16T16:37:37.162Z",
               "cover_image_alt": "Quia voluptates nulla dolorem.",
               "cover_image_thumbnail": "Aut dolores nemo illo.",
               "categories": [
                   {
                       "title": "Course"
                   },
                   {
                       "title": "reading_tracks"
                   }
               ]
           }
       }, {}, {} ]
```


Fig 3.5 POST search/topics

- POST /topic/:slug/courses

```
{
 data: [
    {
            "id": "258",
            "type": "courses",
            "attributes": {
                "title": "Aldor",
                "slug": "Course-Slug-5",
                "description": "Molestiae natus officia. Sunt velit
                "published_at": "2022-05-16T16:37:37.000Z",
                "created_at": "2022-05-16T16:37:37.248Z",
                "updated_at": "2022-05-16T16:37:37.248Z",
                "cover_image_alt": "Maxime ipsa voluptas debitis.",
                "cover_image_thumbnail": "Non voluptatem eum sed.",
                "video_cover_url": "video-url1",
                "eligible_for_certificate": false,
                "app_exclusive": false,
                "audio_language": "English",
                "instructors_data": [
                    Instructor Object, {}, {}, ...
                ]
    }
  ]
}
```


Fig 3.6 POST /topic/:slug/courses

- POST /search/courses

```
{
    "success": true,
    "error": null,
    "total_count": 17,
    "data": [
      {
          "id": "273",
          "type": "courses",
          "attributes": {
              "title": "Verilog",
              "slug": "Course-Slug-20",
              "description": "Minima perspiciatis molestiae. Mollitia quide
              "published_at": "2022-05-16T16:37:37.000Z",
              "created_at": "2022-05-16T16:37:37.000Z",
              "updated_at": "2022-05-16T16:37:37.000Z",
              "cover_image_alt": "Natus quibusdam aliquam at.",
              "cover_image_thumbnail": "Eaque delectus ea quia.",
              "video_cover_url": null,
              "eligible_for_certificate": true,
              "app_exclusive": true/false,
              "audio_language": "english",
              "modules_count": 2,
              "videos_count": 3,
              "total_entities_count": 3,
              "instructors_data": [Instructor Object, {}, {}, ...]
          }
      },
      {},...
    ]
}
```


Fig 3.7 POST search/courses

- GET course/:slug/modules

```
{
  "success": true,
  "error": null,
  "total_count": 2,
  "data": [
      {
          "id": "276",
          "type": "course_chapters",
          "attributes": {
              "title": "Course Chapter 17",
              "slug": "course-chapter-17",
              "created_at": "2022-05-25T11:07:44.572Z",
              "updated_at": "2022-05-25T11:07:44.572Z",
              "order": 2,
              "cover_image": null,
              "total_time": "1h 30m",
              "entities_data": [
                  {
                    "order": 1,
                    "type": "Video",
                    "title": "MARK-IV (now VISION:BUILDER)",
                    "slug": "mark-iv-now-vision-builder-fbe42b15-2cc0-4845-a
                    "published_at": "2022-05-25T11:07:44.000Z",
                    "duration": "00:45"
                  },
```


Fig 3.8 GET course/:slug/modules

3.8 Logic for Calculation for course progress

The logic for calculating of course progress is very simple. First you have to join the user_course_trackings table which will have all the entities count for every course he/she will be viewing divided and join the chapter_entities to count all the chapter_entities for that particular course having a where condition of the chapter. Below is the detailed example

Course - C++

1. Introduction
    a. what is c++
    b. use case
    c. scopes
    d. history
    e. future
2. Loops in C++
    a. what is loop
    b. for loop
    c. while loop
    d. do while loop
3. If/ else
    a. what is if else
    b. conditions
    c. nested if else
    d. else if

Chapter_counts = 3

Total entity count = 13

progress = Chapter_counts/ Total entity count

```
def query
    relation.joins(:chapter_entities)
        .joins("
         LEFT JOIN user_course_trackings as uct ON uct.chapter_id = chapter_owners.chapter_id
        ")
        .where("uct.course_id = #{options[:course_id]}
         AND uct.user_id = #{options[:user_id]}
         AND chapter_owners.owner_id = #{options[:course_id]}
         AND chapter_owners.owner_type = 'Course'
         ")
        .group('chapters.id')
end


def selector(current_query)
    current_query.select('
      chapters.title as title,
      chapters.slug as slug,
ROUND((COUNT(DISTINCT (CASE when(uct.status = 1) then uct.id end))/COUNT(DISTINCT chapter_entities.id))*100) as module_progress
    ')end
```

3.9 Logic for Calculation for video progress

The logic for calculating video progress is not very different from course progress. Actually we store the duration of video in the database itself and we are constantly polling to the backend from the frontend. we are using Vimeo SDK for sending request to the backend and extending support for the frontend

3.9.1 Vimeo SDK

The Vimeo player SDK is a short (but sweet) JavaScript library. We designed it to be easy to use and even easier to access from HTML. There are only two ingredients:

- The link to the SDK
- An embedded Vimeo player

3.9.2 Accessing the player SDK

The easiest way to access the latest player SDK is to reference it directly from a <script> tag in your HTML. Set the **src** attribute to https://player.vimeo.com/api/player.js.

Some points we considering

- Storing the time_elapsed of video in DB as well as in cookies
- Storing the notes in local storage if he doesn't saves the notes in videos and closes the tab

```
function getCookie(cname) {
  let name = cname + "=";
  let decodedCookie = decodeURIComponent(document.cookie);
  let ca = decodedCookie.split(';');
  for(let i = 0; i <ca.length; i++) {
    let c = ca[i];
    while (c.charAt(0) == ' ') {
      c = c.substring(1);
    }
    if (c.indexOf(name) == 0) {
      return c.substring(name.length, c.length);
    }
  }
  return "";
}
```

```
var iframe = document.querySelector('iframe');
var player = new Vimeo.Player(iframe);
```

```
// When the player is ready, add listener for playProgress
player.on('play', function() {
    player.setCurrentTime(getCookie("timeElapsed")).then(function(seconds) {}).catch(function(error) {
        switch(error.name) {
            case 'RangeError':
                // The time is less than 0 or greater than the video's duration
                break;
            default:
                // Some other error occurred
                break;
        }
    });
});
player.getVolume().then(function(playbackRate) {
    console.log(playbackRate);
});
```

```
player.on('timeupdate', function(data) {
    document.cookie = `timeElapsed=${data.seconds}`;
});

player.on('ended', function() {
    alert('Video play completed');
});
```

3.9.3 Calculation for Video Progress

we are sending all the time_elapsed from the above code by constantly polling to the backend

so we just need to calculate the progress by

progress_percentage = time_elpased/duration * 100

After that we are saving the progress_percentage and status for this particular video in user_course_trackings table

we will be checking 2 conditions majorly for that

1. when time_elapsed > duration

   Then the progress percentage will be 0 and status will be “pending”

2. when progress percentage > 90 %

   will be setting status to complete

3. if a user rewatches a video

   his highest progress will be taken and once status is “completed” then it will not change

We are sending progress_percentage as a hash that will contain status and progress percentage after calculation for that particular video.

BASIC CODE STRUCTURE

```
def progress_metrics(time_elapsed:, entity_id: )



   status = @model.status

   progress_percentage = @model.progress_percentage



   progress = Video.find_by(id: entity_id).progress_tracker.progress_percentage(time_elapsed: time_elapsed)

   progress_hash = {

    progress_percentage: [progress, progress_percentage].max,

    status: status

   }



   if progress_hash[:progress_percentage] >= 90.0 && progress_hash[:status] == 'pending'

    progress_hash[:status] = 'completed'

   end



   progress_hash

end
```

4. Performance Analysis

1. **Seamless integration of new features in the existing Repository**

   Web developers appreciate Rails for their easy-to-navigate feature across various web forums. This feature is supported by the Rails website migration. The Functional Model that highlights the default Rails Active_record website can be easily extracted easily with the difference between different SQL backgrounds. Instead of writing a schema in pure SQL language, we can use the transport feature that allows us to use the simple Ruby DSL syntax to define changes to tables and records. As a result, RoR allows for the creation of a database-agnostic schema and models that facilitate the migration of Rails applications to different sites.

2. **Traffic for the Website increased**

   As new features and interesting content was introduced to the existing site, user activity and engagement have increased significantly.

3. **Calculation of progress**

   Algorithmic approach that was discussed above was accurate up to .2 decimals and retrieving the progress was efficient because using indexes in the user_course_trackings table.

4. **Writing test cases using RSpec**

   RSpec is a Ruby system language testing module. RSpec is different from standard xUnit frameworks like JUnit because it is a behavior-oriented development tool. This means that RSpec written tests focus on the "behavior" of the test application. RSpec focuses on how the application works, not on how it works, that is, what the application actually does.

   Running RSpec on the existing code resulted in finding and eliminating any chance of bugs that would cause unnecessary behaviors

5. Conclusions

The title of the thesis is “Development of Online Course Application with Ruby on Rails framework”. This report described how the MVC architecture applies to the Web Community application under the Ruby on Rails framework environment. Using the RoR technology, the application has been divided into smaller components. Developing application functionality into modules has helped to separate stable code from frequently changed one, to reuse source code, and to maintain and extend the application more easily.

The main objectives at the start of the thesis were as follows

- In designing and developing a data structure that can comprehensively define and hold each course's properties.

- To design and develop a friendly user interface by which users of the system can interact with it in a browser environment.

- The users must be able to see course contents.

- The users must be able to track their course progress.

- The users must be able to resume from the chapter where he leaves.

- The users must be able to find a particular course with name or title.

- The users must be able to take notes in between the courses.

At the end of the thesis the following things are achieved

- A front-end is developed for the client browser which is a system by which users can interact with the application.

- A data structure that comprehensively defines and stores course details

- Once the user has logged in, he is able to see the progress of individual video/ course/ chapters

- Once the user has logged in, he is able to find all the courses he has been registered

- Once the user has logged in, he is able to resume the course from where he left

5.1 Future Scope

Further work involves continuing writing and executing tests, solving different errors and application bugs, implementing unimplemented features. After securing and testing are finished, the application will be brought to the production environment and will be deployed on the server

Appendix A - References

1. Dave Thomas, David Heinemeier Hansson: Agile Web Development with Rails Second Edition, Pragmatic Bookshelf, ISBN 0-9776166-3-0

2. Dave Thomas, Chad Fowler and Andy Hunt: Programming Ruby Second Edition, Pragmatic Bookshelf, ISBN 0-9745140-5-5

3. Andre Lewis, Michael Purvis, Jeffrey Sambells and Cameron Turner: Beginning Google Maps Applications with Rails and AJAX, Apress, ISBN 1-59059-787-7

4. Andy Budd, Cameron Moll and Simon Collison: CSS Mastery, Friendsof, ISBN 1-59059-614-5

5. Ruby Documentation http://www.ruby-lang.org/en/

6. ROR Documentation http://rubyonrails.org/

7. http://thepaisano.files.wordpress.com/2008/04/rails2.png

8. http://rewrite.rickbradley.com/pages/moving_to_rails/